Snakes and Trees

Walter Brown
Marc Paterno
William Tanenbaum

Contents

Lions and tigers and bears, oh my!
— DOROTHY GALE

1 Introduction

Fermilab has been asked to take on a project having to do with making access to COBRA data from ROOT simpler. However, the description of the project has not seemed clear to us. We have written this document to put forth our understanding of the contemplated project.

1.1 Desiderata

We have excerpted the following desiderata (for a ROOT/COBRA plug-in) from a thread of e-mail correspondence to which Jim Branson, Vincenzo Innocente, and Paris Sphicas (among others) have contributed.

1. We want to read existing COBRA data.
2. We want to train people to use the COBRA data model.
3. We want it to be easy to produce distributions from the data.
4. It should be easier to use than the full COBRA framework, but need not be as fast as reading ROOT ntuples.
5. Export COBRA and ORCA high-level objects to ROOT, without exposing the data as ROOT *TTree* instances.
6. We want a plug-in to "make COBRA/ORCA objects readable at the ROOT prompt."

We do not believe a single coherent product can meet all these desiderata simultaneously.

1.2 Outline

In this document, we sketch our proposal for a ROOT/COBRA interface. In order to devise such a proposal, one must first have in mind clear goals and requirements. To this end, we first discuss the reason for the creation of this project, and the perceived problems it is to solve. We also touch briefly upon problems this project is *not* intended to solve.

2 Why is Another Interface for COBRA Wanted?

What is the cause of the desire for another interface to COBRA? There is a perception among some CMS collaborators that the COBRA framework is too complicated for many users. Users who wish to produce histograms for analysis tend to avoid direct use of the COBRA framework for some (or all) of the following reasons:

1. They do not understand the mechanisms for compiling and linking framework programs.
2. They do not understand how to navigate the web of reconstructed objects.
3. They do not understand the individual reconstructed objects, and in any case view them as too complex.
4. They find the edit/compile/link/run cycle of the framework is too slow.
5. They prefer to work in an interactive (rather than batch) processing environment.
6. They do not understand the mechanisms for specifying the data to be read.

Since the most commonly-used interactive environment in CMS is ROOT, we have been asked to "make the CMS event data available in ROOT," with the goal of eliminating—or at least reducing—the perceived difficulties listed above.

Making the CMS event data available in ROOT could mean many things. Our understanding is that we have *not* been asked to devise a ROOT tuple suitable for all analysis purposes.[1] We believe instead that we have been asked to make it possible for users unfamiliar with the COBRA framework to work at the ROOT prompt, reading existing CMS data in their current native framework, and making plots of the reconstructed quantities available from those data. We are furthermore constrained to work within the bounds of what can be done in several weeks' time, which precludes significant development of any of the CMS infrastructure software (POOL, ROOT, COBRA, SEAL, *etc.*).

3 Requirements through Examples of Use

There is insufficient time to produce a detailed requirements document for the ROOT/COBRA interface. We propose instead to describe "requirements" by specifying *examples of use*, that is, tasks we propose the user be able to perform at the ROOT prompt. This includes some parts of the interface of the classes discussed. In all cases, these proposals are *very* preliminary; it should be expected for many to change, as requirements are better understood and as constraints imposed by the underlying software are discovered.

[1] It is also our belief that this would have been an unrealistic goal. No single tuple design would be suitable for all analysis uses.

3.1 General "Philosophy"

In section 2 we presented the reasons for producing a ROOT/COBRA interface. They lead us to the following guidelines:

1. It must be easy to specify the data to be used.
2. No compilation and linking (except as provided by ACLiC) should be necessary.[2]
3. Simple operations should require no more than use of ROOT's `TTree::Draw` command.
4. The user must *not* be required to run a COBRA framework program to transform the data for use in ROOT.

One of the suggested goals for this project is to introduce users to the CMS event-data model (EDM) and the classes therein. We therefore do *not* attempt to translate reconstruction objects to another format, but instead concentrate on making access to these objects within a ROOT session as easy as reasonably possible.

We note that there is a difference in design goals between *reconstruction* objects and *analysis* objects. Reconstruction objects are, in a way, akin to database data in normal form. They involve complex relationships, arranged to avoid redundancy in order to obtain *correctness* by design. The relationships are designed to make it impossible for data to become inconsistent. Duplicated data are (or should be) rare. Complex transformations (reconstruction and subsequent corrections) are the reason for the existence of these objects, and so correctness is paramount. In contrast, analysis objects are more like unnormalized database data, and they exist for the same reasons one sometimes allows unnormalized data in a database:

- for greater speed of access, or
- to make use of the data less complex.

Complex transformations should rarely be done on analysis objects, and so the benefits of "normal form" are less important than ease of use. CMS document CD-doc-435 presents much more information on the differences between reconstruction and analysis objects. A natural method (as discussed in CD-doc-435) of providing analysis objects would be to transform COBRA format data into user-specified ROOT tuples. Our understanding is that this solution is explicitly rejected by CMS at this time.

In the following sections, we propose a design we believe represents a reasonable compromise between the various competing requirements and constraints discussed above.

3.2 Design Overview

We propose to create a facility to allow the reading of COBRA data, from within a ROOT session, and which provides automatic reorganization of a subset of the COBRA data into the ROOT format most useful for use from the ROOT prompt. To this end, we propose the creation of a class, *TCobra*, which has the responsibility of creating ROOT-format data from the COBRA data.[3] This class would coordinate reading of the COBRA-format data, creation, filling, and management of *TTree*s to contain the data, and all associated ROOT "housekeeping" necessary for this to work.

---

[2] ACLiC is ROOT's "Automatic Compiler of Libraries for CINT," which uses the same C++ compiler used to build ROOT to build dynamic link libraries containing arbitrary C++ code which may then be called from the ROOT prompt.

[3] We recognize that COBRA data files *are* ROOT files. However, these ROOT files are organized in a fashion that is useful for reconstruction, but that is *not* convenient for use from the ROOT prompt. The organization imposed by the transformation done by *TCobra* provides this convenience of use.

In sections 3.3–3.6 we sketch the important operations of *TCobra*. Section 3.7 shows some uses of the resulting *TTree*, section 3.8 notes some caveats, and section 3.9 mentions additional functionality that could be added in a later release, if such enhancements seem warranted.

3.3 Connecting to COBRA Data

During creation of a *TCobra* object, the user specifies the data to be read. Construction of the *TCobra* object does *not* cause the data to be read; reading is delayed until a later step in the use of the object. It is possible to create multiple *TCobra* objects, but it is *not* possible to copy (by copy construction or copy assignment, or cloning) a *TCobra* object. This is disallowed because of the cost of copying the underlying ROOT objects; unintentional copying would likely cause terrible performance problems. If, for some reason, a user wishes to create two identical *TCobra* objects, it is always possible to construct two such objects independently, using the same specification of input data.

```
1 // Specify the data to be read. Only system input collection
2 // specifiers shall be supported in the initial version.
3 TCobra c("/System/aaa/bbb/ccc");
```

3.4 Creating a *TTree*

Each *TCobra* object is limited to creation of a single *TTree* instance. Supporting use of more than one instance would increase the complexity of the user interface, with no important gain in functionality that we can identify.

The *TTree* instance is associated with a given (operating system) file and with a *TFile* object, which is used as a backing store. When the *TCobra* object is destroyed (at the end of the ROOT session, if not before), the operating system file remains.

```
1 // Specify the name of the TTree to be created, and the name of
2 // the file that shall be used as its backing store.
3 c.createTree("mytree", "myfile");
```

3.5 Specifying the Objects to be Accessed

The user specifies the objects to be written to the *TTree* instance by calling `createBranch` once for each object to be written. The user specifies:

1. the name of the *RecAlgorithm* whose output is wanted, and
2. the name of the branch to which the output shall be written.

The *RecCollection* produced by the specified *RecAlgorithm* shall be “unwound” into its separate components, which shall be written into a *TClonesArray* on the branch. The call to `createBranch` does not cause reading of the input data.

Following the default behavior of the COBRA framework, if the user requests an EDM object which is *not* in the specified data, reconstruction on demand will be used to provide the requested object. In the first release, only the default version of the algorithms will be used, and no control over reconstruction on demand will be available.

```
// Specify which EDM objects are to be read (or created),
// and the name of the branch to which it shall be written.
c.createBranch("CombinatorialTrackFinder", "ctf");
c.createBranch("GlobalMuonTrackFinder", "gmtf");
```

3.6 Filling the *TTree*

After the user has specified what branches are to created, he arranged to fill the branches:

```
// Fill the TTree. Only one of the following should be called.
c.fill();    // to read all events, or
c.fill(100); // to specify that only 100 events are to be read
```

This is the time at which the data is read, and so this step may be very time-consuming.

3.7 Using the Result

After the above calls have been executed, the user has available a ROOT *TTree* instance, with branches containing *TClonesArray*s of the requested EDM objects. The user is able to do whatever ROOT allows to be done with such a *TTree* and its contents.

To help illustrate what will then be possible, we provide several examples. In these examples, we assume `mytree` is a *TTree* containing two branches: the branch `jet` contains jets from some algorithm, and the branch `ele` contains electrons from some algorithm. We also assume that the entries of the *TClonesArray*s have been sorted on `pt`, the transverse momentum of the object.

To obtain a plot of the transverse momentum of *every* jet:

```
root>  mytree.Draw("jet.pt()")
```

To obtain a plot of the transverse momentum of the leading jet in each event:

```
root> mytree.Draw("jet[0].pt()")
```

Note that if some event contained no jets, `TTree::Draw` is smart enough to skip that event.

To obtain a scatter-plot of the transverse momenta of the two leading jets in each event:

```
root> mytree.Draw("jet[0].pt():jet[1].pt()")
```

To obtain a plot of the transverse momentum of the leading electron in each event for which the leading jet had `pt() > 20.0`:

```
root> mytree.Draw("ele.pt()","jet[0].pt()>20.0")
```

For calculations that require looping constructs more complicated than looping over all entries in a *TClonesArray*, the user will generally have to write a CINT macro or C++ code.

3.8 Additional Caveats

Because the time allocated for implementation is short, and because there may be as-yet-unknown technical challenges ahead, we specify some of our assumptions here. If these assumptions turn out to be untrue, it is likely that the project will *not* be possible within the allocated time.

- We assume it is straightforward to create the ROOT dictionaries for all classes for which interactive use is desired.
- We assume there is no problem in having both ROOT and SEAL dictionaries for the same classes in the same program.
- We assume it is straightforward to write the EDM objects into a *TTree*, since these data are currently written in a ROOT format (although that format is different).
- Persistent POOL links between objects are unlikely to be traversable until such time as support for traversing links to objects in *TTree*s is added; adding such support is outside the scope of this project.

We do not know what fraction of the inter-object navigability will be available from within ROOT. The time budget allocated precludes significant modification of the COBRA classes to support use within interactive ROOT.

It is likely that the memory demands of this use of ROOT will be quite large. Code is required for each of the objects to be used; much of this code is required again in the ROOT dictionaries, for interactive use; some code is required a third time in the SEAL dictionaries, for I/O purposes.

Reading data files with this software shall be no faster than when done with the COBRA framework; the same technology is being used. Looping over events in the created *TTree* may be significantly faster. The ROOT files created may be very large, because no selective pruning of data is done, except that resulting from the user's choice of branches to be written.

3.9 Possible Later Additions

If user interest dictates, a subsequent version of this software may offer additional functionality. Some enhancements to be considered are:

- more user control over the EDM objects selected, and
- user control over whether reconstruction-on-demand is done.

4 Design Alternatives

The design proposed above is a "middle ground" between two other alternatives. These alternatives, and the reasons we rejected them, are described below.

4.1 ROOT *TTree*s of Analysis Objects

One solution that we believe would be popular with many physicists would be to create ROOT tuples, consisting of *TTree*s filled with analysis objects. A related option would be to create *TTree*s filled with `struct`s analogous to analysis objects. While we believe that, in the long run, such data formats are likely to appear, the design of such tuples must be guided by the physics tasks for which they are intended. We do not believe that a single design can fit all needs, and it is clearly not reasonable for us to propose such a design. This solution appears to be incompatible with the CMS vision of transparent access to all previous "levels" of reconstruction at all times. Only time will tell whether this vision is shared by the majority of CMS collaborators performing analysis.

4.2 A ROOT-based Analysis Framework

A second alternative is to create what might be called a ROOT-based analysis framework. By this we mean providing a class that would open and read COBRA-format data files, that would provide an event loop, and that would allow a user-defined function to be called on each event in the sequence of events. The user code would be given access to the event, and would use the standard COBRA EDM mechanisms to access elements in the event.

This solution meets many of the same desiderata as does the chosen solution:

- users do not have to understand how to build a framework program;
- the edit/compile/link/run cycle is improved to the same degree as for the chosen solution;
- it allows users to work in the ROOT interactive environment, since ROOT histograms, *etc.*, that are created in the user code would be available in the same interactive ROOT session;
- it provides the same easier specification of input collections.

In addition, it meets one of the desiderata *not* met by the chosen solution: it provides a less-difficult introduction to use of the COBRA EDM than does direct use of the COBRA framework. Additionally, this solution seems more general-purpose, since it provides full navigation between all EDM objects. Finally, code developed for such a user-function may be directly transported to the COBRA framework (and thus used in ORCA).

However, this solution does *not* meet one of the desiderata which is met by the chosen solution: it does not shield the user from the complexities of navigation of the web of reconstructed objects. We stress that this goal is in conflict with those of the paragraph above and can see no way to meet it and those simultaneously. Our understanding is that the ease of use is more important than the other advantages of this alternative solution. If this understanding is incorrect, then this solution should be re-evaluated.